

Principles of Natural RPC

This section covers the following topics:

- General Information
 - Natural RPC Operation in Non-Conversational Mode
 - Natural RPC Operation in Conversational Mode
 - Conversational versus Non-Conversational Mode
 - Database Transactions
 - Restrictions and Limitations when Using Natural RPC
-

General Information

The following topics are covered below:

- Purpose
- Advantages of Natural Remote Procedure Calls
- Natural RPC Modes of Operation
- Availability on Various Platforms
- Support of Non-Natural Environments
- Prerequisites

Purpose

The Natural RPC facility enables a client Natural program to issue a CALLNAT statement to invoke a subprogram in a server Natural. The Natural client and server sessions may run on the same or on a different computer.

Example:

A Natural client program on a Windows computer can issue a CALLNAT against a mainframe server in order to retrieve data from a mainframe database. The same Windows computer can act as a server if a Natural client program running under, for example, OpenVMS issues a CALLNAT requesting data from a server Natural.

Advantages of Natural Remote Procedure Calls

Natural RPC exploits the advantages of client server computing. In a typical scenario, Natural on a Windows client computer accesses server data (using a middleware layer) from a Natural on a mainframe computer. The following advantages arise from that:

- The end user on the client can use a Natural application with a graphical user interface.
- A large database can be accessed on a mainframe server.
- Network traffic can be minimized when only relevant data are sent from client to server and back.

Natural RPC Modes of Operation

The Natural Remote Procedure Call offers the following modes of operation:

- non-conversational mode (in the following texts this mode is meant unless otherwise specified)
- conversational mode

These modes are described in detail in the following sections. For a comparison of the advantages and disadvantages of these modes refer to Conversational versus Non-Conversational Mode.

Availability on Various Platforms

You can use the Natural RPC on various platforms under the following operating systems:

Mainframe Environments

- OS/390
- VSE/ESA
- VM/CMS
- BS2000/OSD

Natural RPC on mainframes is supported under the following TP monitors:

- Com-plete
- CICS
- IMS/TM
- TSO
- UTM

Also, it is available in batch mode.

Other Environments

- OpenVMS
- UNIX
- Windows

On all of these platforms, Natural can act as both client and server.

Exception: Under Windows 98 and Windows ME, Natural can only act as client.

Support of Non-Natural Environments

Non-Natural environments (3GL and other programming languages) are supported on the client and the server side. Thus, a non-Natural client can communicate with a Natural RPC server, and a Natural client can communicate with a non-Natural RPC server. This is enabled by the use of the EntireX SDK.

Prerequisites

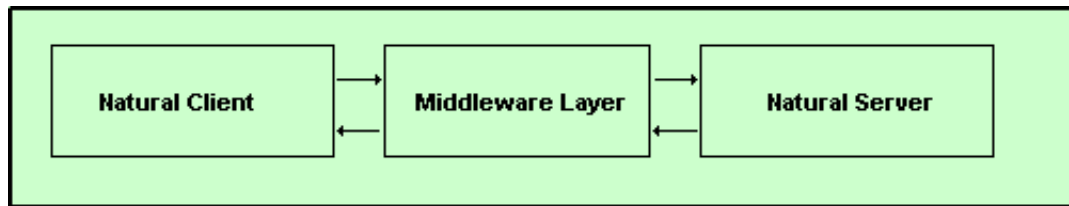
The Natural RPC interface requires the following products:

- Software AG EntireX Broker (including the stubs).
- Software AG Entire Net-work (if the transport method used by EntireX Broker is Entire Net-work)
- TCP/IP (if the transport method used by EntireX Broker is TCP/IP)
- EntireX SDK for non-Natural programming language support.
- Directory services if the location transparency provided by Software AG EntireX Broker is used.

Natural RPC Operation in Non-Conversational Mode

The non-conversational mode should be used only to accomplish a single exchange of data with a partner. See also Conversational versus Non-Conversational Mode.

The Natural RPC technique uses the Natural statement CALLNAT, so that both local and remote subprogram calls can be issued in parallel. Remote program calls work synchronously. As a remote procedure call, a CALLNAT would, simply speaking, take the following route:



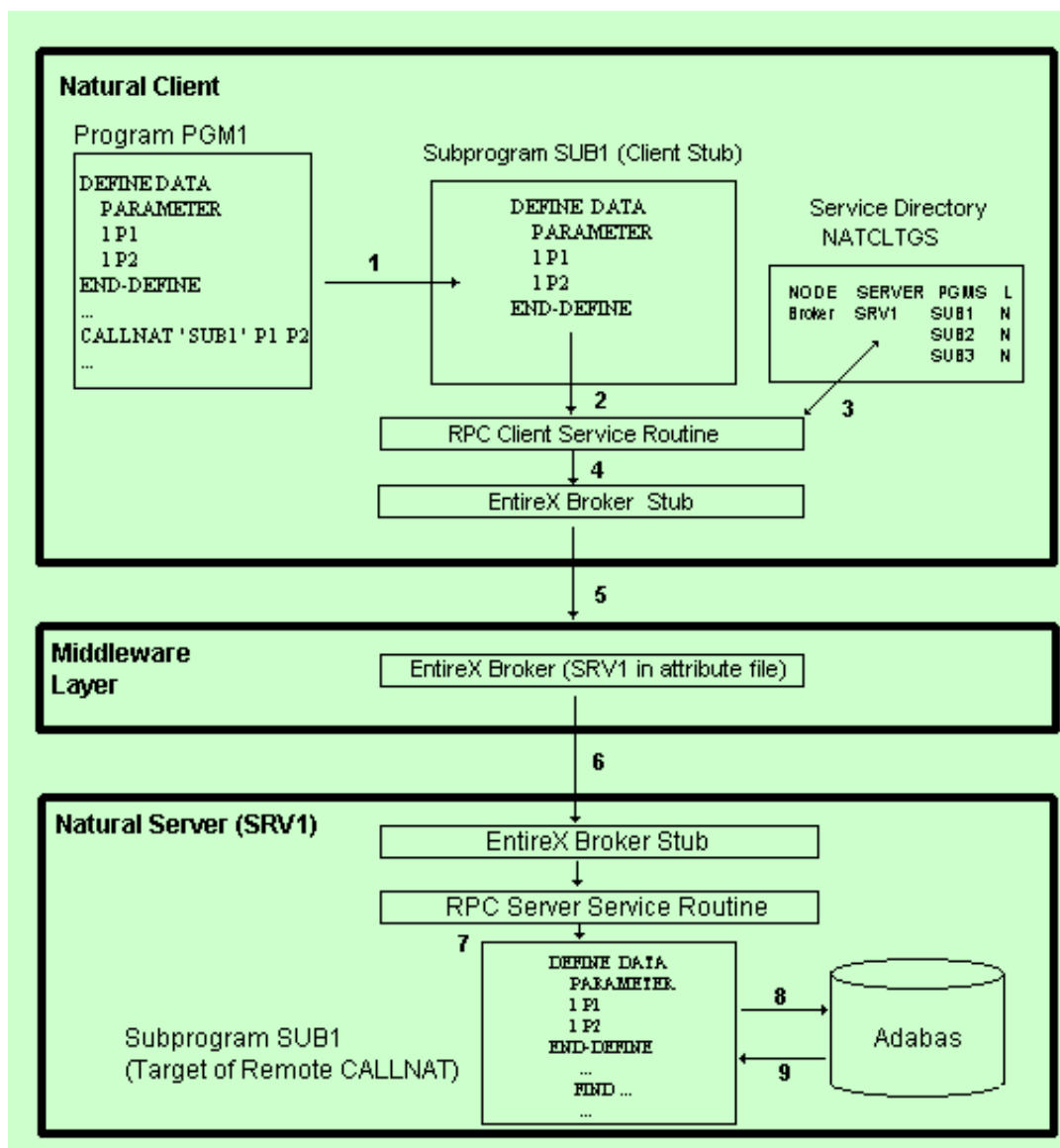
The CALLNAT issued from the Natural Client is routed via a middleware layer to the Natural Server which passes data back to the client.

Usually, the middleware layer consists of the Software AG product EntireX Broker which uses the ACI protocol. EntireX Broker uses either NETWORK or TCP/IP as communication layer.

A detailed example of the RPC control flow is described below.

Issuing CALLNATs in an RPC Environment

CALLNAT control flow details in a remote procedure are illustrated below. For greater clarity, the return path is not shown, but it is analogous; the numbers refer to the description:



1. From the Natural client, the program PGM1 issues a CALLNAT to the subprogram SUB1. PGM1 does not know if its CALLNAT will result in a local or in a remote CALLNAT.
As the target SUB1 resides on a server, the CALLNAT accesses a "stub" subprogram SUB1 instead. This client stub subprogram has been created automatically or by using the SYSRPC Utility's Stub Generation (SG) function.

The stub has the same name as the target subprogram and contains parameters identical with those used in program PGM1 and the target subprogram SUB1 on the server. It also contains control information used internally by the RPC.

If the parameter AUTORPC is set to ON and Natural cannot find the subprogram in the local environment, Natural will interpret this as a remote procedure call and will generate the parameter area dynamically during runtime.

It will also try to find this subprogram in the Service Directory.

For more information on the SYSRPC Stub Generation function, see also *Creating Stub Subprograms*.

If you want to work without stubs, see also *Working with Automatic Natural RPC Execution*.

2. The stub then sets up a CALLNAT to an RPC client service routine.
3. The client RPC runtime checks in the service directory NATCLTGS on which node and server the CALLNAT is to be performed and whether a logon is required.
The CALLNAT data including the parameter list and optionally the logon data are passed to a middleware layer.
4. In this example, this middleware layer consists of the Software AG product EntireX Broker. Therefore, the CALLNAT data is first passed to an EntireX Broker stub on the client.
5. From the EntireX Broker stub, the CALLNAT data is passed to the EntireX Broker. The EntireX Broker is a product that can reside:
 - on the client computer
 - on the server computer or
 - on a third platform.

For the data to be passed on successfully, the server SRV1 must be defined in the EntireX Broker attribute file and SRV1 must be already up, thus having registered with EntireX Broker.

For information on how to define servers in the EntireX Broker attribute file, see the EntireX Broker documentation.

6. From the middleware layer, the CALLNAT data is passed on to the EntireX Broker Stub on the Natural Server platform and from there to the RPC server runtime.
The server runtime validates the logon data (if present) and performs a logon (if requested).
7. The RPC server runtime invokes the target subprogram SUB1 and passes the data, if requested.
At this point, the target subprogram SUB1 has all the required data to execute just as if it had been invoked by a local program PGM1.
8. Then, for example, SUB1 can issue a FIND statement to the server's Adabas database. SUB1 does not know whether it has been performed by a local or by a remote CALLNAT.
9. Adabas FINDs the data and passes them to SUB1.
Then, SUB1 returns the Adabas data to the calling server service routine, which passes it back to PGM1 via the middleware layer using the same route as described in Steps 1 to 8, but in reverse order.

Natural RPC Operation in Conversational Mode

A conversational RPC is a static connection of limited duration between a client and a server. It provides a number of services (subprograms) defined by the client, which are all executed within one process that is exclusively available to the client for the duration of the conversation.

Multiple connections (conversations) can exist at the same time. They are maintained by the client by means of conversation IDs, and each of them is performed on a different server. Remote procedure calls which do not belong to a given conversation are executed on a different server, within a different process.

During a conversation, you can define and share a data area called context area between the remote subprograms on the server side.

A conversation may be local or remote.

Example:

```
OPEN CONVERSATION USING SUBPROGRAM 'S1' 'S2'
    CALLNAT 'S1' PARMS1
    CALLNAT 'S2' PARMS2
CLOSE CONVERSATION ALL
```

Both subprograms (S1 and S2) must be accessed at the same location, i.e. either locally or remotely. You are not allowed to mix up local and remote CALLNATs within a conversation. If the subprograms are executed remotely, both subprograms will be executed by the same server replicate.

Analogously to non-conversational RPC CALLNATs, conversations may first be written and tested locally and can then be transferred to the servers.

General Rules for Local/Remote Subprogram Execution

Local Subprogram Execution

If you execute subprograms locally, the following rule applies:

- A subprogram may not call another subprogram which is a member of the conversation.

Other subprograms not listed in the OPEN CONVERSATION statement may be called. They are executed in non-conversational mode.

Remote Subprogram Execution

If you execute subprograms remotely, the following rule applies:

- A subprogram S1 may call another subprogram S2 which is a member of the conversation.

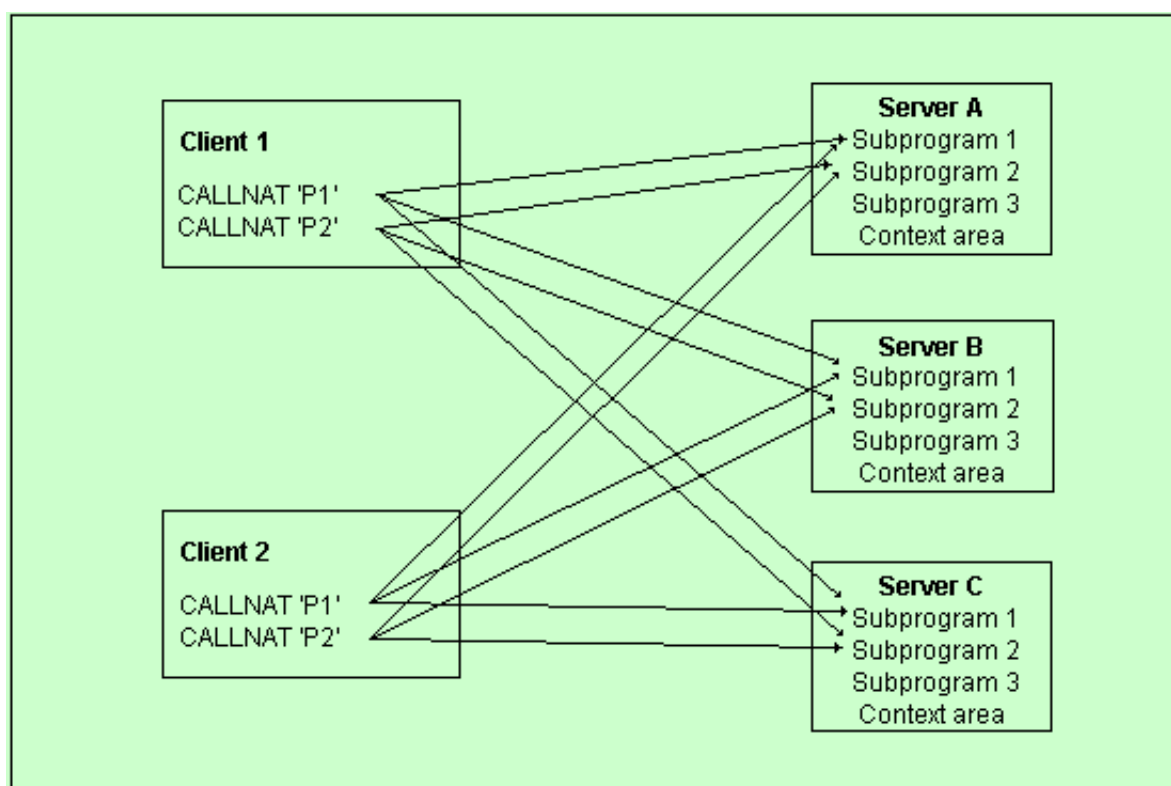
This CALLNAT will be executed in non-conversational mode because it was invoked indirectly. Thus, the subprogram S2 does not have access to the context area.

Conversational versus Non-Conversational Mode

In a client-server environment where several clients access several servers in non-conversational mode, there may be the problem that identical CALLNAT requests from different clients are executed on the same server.

This means, for example, that a CALLNAT 'P1' from Client 1 executes Subprogram 1 on server A (P1 is writing a record to the database). The transaction for Client 1 is not yet complete (no END TRANSACTION) when Client 2 also sends a CALLNAT 'P1' to server A, thus overwriting the data from Client 1. If Client 1 then sends a CALLNAT 'P2' (meaning END TRANSACTION), Client 1 thinks its data have been saved correctly while the data from client 2's identical CALLNAT have in fact been saved.

The diagram below illustrates this with two clients and three servers. In such a scenario, you cannot control whether two identical CALLNATs from two different clients access the same subprogram on the same server:



CALLNAT 'P1' from Client 1 can access Subprogram 1 on server A, B, or C. CALLNAT 'P1' from Client 2 has the same choice. It is obvious that interference can be a problem here if the subprograms are designed to be executed within one process context.

You can avoid the potential problems of a non-conversational RPC by defining a more complex RPC transaction. You do this by opening a conversation, for example, on Client 1 comprising CALLNAT 'P1' and CALLNAT 'P2'. Opening such a conversation reserves one entire server replicate (for example, server A) and no other remote CALLNATs may interrupt this conversation on this server until the conversation is closed.

General Rules for Use of Conversational/Non-Conversational RPC

As a general rule, the following applies:

- Use the **conversational RPC** to ensure that a defined list of subprograms is executed exclusively within one context.
- Use the **non-conversational RPC** if each of your subprograms can be used within a different process or if the transaction does not extend over more than one server call. The advantage of this is that no server blocks over a significant amount of time and you only need a relatively small number of server replicates.

Possible Disadvantage of Using Conversational RPC

A possible disadvantage of conversational RPCs is that you reserve an entire server replicate, thus blocking all other subprograms on this server. As a consequence, other CALLNATs might have to wait or more server replicates must be started.

Database Transactions

The database transactions on the client and server side run independent of each other. That is, an END TRANSACTION or BACKOUT TRANSACTION executed on the server side does not effect the database transaction on the client side and vice-versa.

At the end of each non-conversational CALLNAT and at the end of each conversation, an implicit BACKOUT TRANSACTION is executed on the server side. To commit the changes made by the remote CALLNAT(s), you have the following options:

- Non-conversational CALLNAT
- Conversational CALLNAT

Non-conversational CALLNAT

1. Execute an explicit END TRANSACTION before leaving the CALLNAT.
2. Set the Natural profile parameter ETEOP to ON (mainframe platforms) or set the Natural profile parameter OPRB to OFF (non-mainframe platforms). These settings result in an implicit END TRANSACTION at the end of each non-conversational CALLNAT.

Conversational CALLNAT

1. Execute an explicit END TRANSACTION on the server before the conversation is terminated by the client
2. Set the Natural profile parameter ETEOP to ON (mainframe platforms) or set the Natural profile parameter OPRB to OFF (non-mainframe platforms). These settings result in an implicit END TRANSACTION at the end of each conversation.
3. Before executing the CLOSE CONVERSATION statement, call the interface USR2032N on the client side. This will cause an implicit END TRANSACTION at the end of the individual conversation.

Restrictions and Limitations when Using Natural RPC

When executing a subprogram by using the Natural RPC facility, several differences to local execution apply.

- User Context Transfer
- System Variable Transfer
- Parameter Handling in Error Situations
- Natural Statement Reactions
- Dynamic Arrays in Subprograms
- Location of Conversations
- Future Restrictions of Statement Usage with RPC

User Context Transfer

Excepting the user identification, no user context is transferred to the server session, for example:

- all client session parameters remain unchanged and do not affect the execution on the server side;
- open transactions on the client side cannot be closed by the server and vice versa;
- client report handling and work-file processing cannot be continued on the server side and vice versa;
- the handling of the Natural stack cannot be continued either.

System Variable Transfer

No system variables except *USER can be transferred from the client to the server side.

Parameter Handling in Error Situations

Parameter handling in error situations is different:

- If an error occurs during local execution, all parameter modifications performed so far are in effect, because parameters are passed via "call by reference".
- If an error occurs during remote execution, however, all parameters remain unchanged.

Natural Statement Reactions

Several Natural statements may react in a different way, for example:

Statement	Description
OPEN/CLOSE CONVERSATION	If executed on a server, these statements do not affect the client session. When the server itself acts as a client for another server (as agent), these statements only affect the conversations on the second server.
PASSW	The password setting remains active at the server side only, also for subsequent executions by other users.
SET CONTROL, SET GLOBALS, SET KEY, SET TIME, SET WINDOW	No settings are returned to the caller.
STACK	All stack data are released after execution.
STOP, TERMINATE	These statements do not stop the client session.

Dynamic Arrays in Subprograms

Dynamic arrays in subprograms can only be handled if you do not execute the remote CALLNAT via a stub.

Location of Conversations

Both subprograms (S1 and S2) must be accessed at the same location, i.e. either locally or remotely. You are not allowed to mix up local and remote CALLNATs within a conversation. If the subprograms are executed remotely, both subprograms will be executed by the same server replicate.

Future Restrictions of Statement Usage with RPC

Current State

With Natural Version 3.1 (mainframe environments) or Version 4.1 (Windows, OpenVMS and UNIX environments), the use of the following statements in conjunction with RPC is theoretically possible, but not recommended, as it causes undesired effects:

Statement	Description
TERMINATE	Using this statement causes the server to be terminated, regardless of conversations that may still be open.
FETCH, RUN, STOP	Using these statements causes the CALLNAT context to be lost. Upon a FETCH, RUN or STOP statement, the server detects that it has lost its CALLNAT context and returns a corresponding Natural error message to the client; at that time, however, the statement has already been executed by the server. Exception: This does not apply to FETCH RETURN.
INPUT	Input values are unpredictable when the input data are read from a file (and not from the stack).

Future State

The use of the statements FETCH, INPUT and RUN in conjunction with the Natural RPC will be inhibited.

Statement	Description
FETCH, RUN, INPUT	Not permitted.
STOP, TERMINATE	Same as ESCAPE ROUTINE.